RSA

RSA algorithm is based on factoring assumption which states that,

Let genmodulus be a polynomial time algorithm that, outputs (N, p, q) where N = pq, and p, q are n=bit primes except negligible probability. Then, we define an experiment such that:

1) Run GenModulus to obtain (N, p, q)
2) Adversary is given N, and outputs p', q' > 1
3) The output is defined to be 1 if p'.q' = N, 0 otherwise.

We say that factoring is hard for all probabilistic polynomial time adversaries and that there exists a negligible function negl such that:

Pr [Factor (n) = 1] <= negl(n)

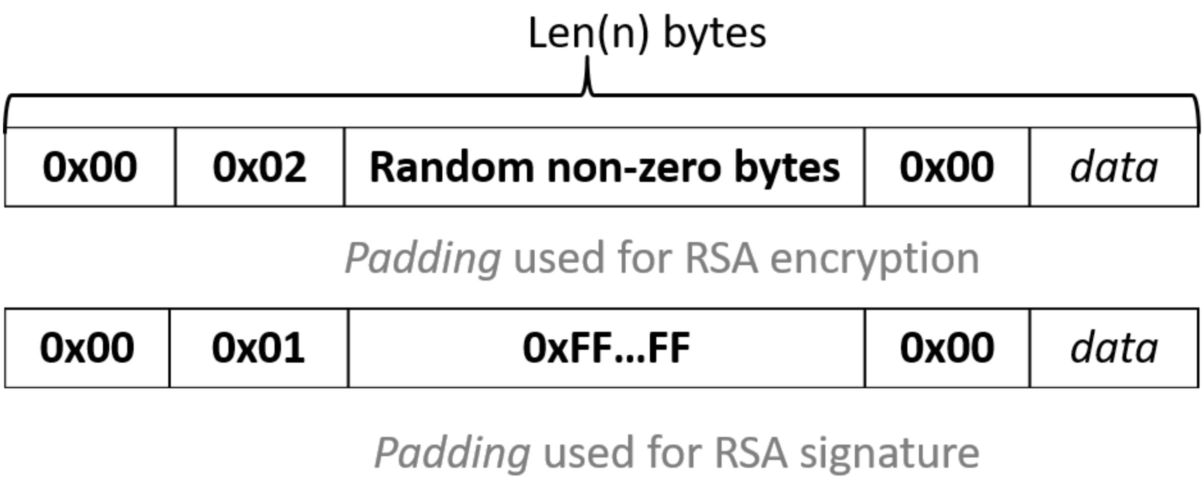
RSA can now be defined as, given N, e, y find x such that x ^e = y mod N.

RSA experiment,

1) Run GenRSA() to obtain (N, e, d)
2) Choose y ← Z*
3) Adversary is given N, e, y and outputs x belonging to Z*
4) Output of the experiment is 1 if x ^e = y mod N. 0 otherwise.

Therefore, we say that RSA problem is hard if for all probabilistic polynomial time algorithms, there exists a negligible function negl such that

Pr[RSA-invA;GenRSA(n) = 1] ≤ negl(n)

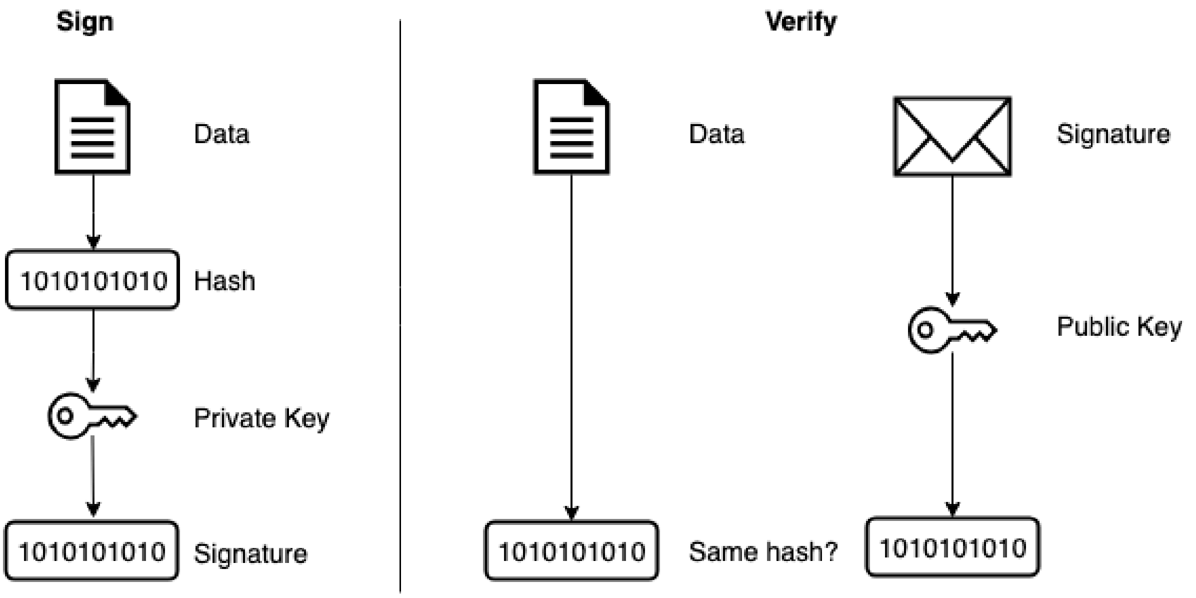


Signature Experiment,

1) Run GenSign() to obtain keys (pk, sk)
2) Adversary is given pk and oracle access to Signsk.
3) Let Q denote the queries whose signatures were requested by Adversary during its execution. The output of the experiment is 1 if m does not belong to Q and Vrfy(m, sig) = 1

A signature scheme is existentially unforgeable under an adaptive chosen-message attack if for all probabilistic polynomial time adversaries, there exists a negl function such that

$$\mathbf{Pr[Sig\text{-}forgecma\ A;\Pi(n) = 1] \le negl(n)}$$



Build Algorithm

1) Generate p, q such that they are n-bit prime numbers.
2) Calculate N = p.q and phi_N = (p-1) * (q-1)
3) Calculate public and private kets by choosing e and d, such that e is co-prime with N and phi_N, similarly, d is chosen such that d * e mod phi_N = 1.
4) Define NULL byte, fixed Bytes as well as the bytes to be padded for padded RSA.
5) Message from the user is taken and converted into a list of ascii values per word.
6) The integer list is then encrypted using public key of the receiver.
7) Convert the encrypted list into binary and pad the bytes. We finally have the cipher we can send over a public channel.
8) Additionally, we now have to calculate hash or the digital signature as its popularly called, we first take the original message and convert it to binary.

9) We use our collision resistant hash function created in previous tasks to calculate the hash value, we then encrypt this hash using our private key.

10) The cipher text along with the hash is then finally sent to receiver, At receiver's end, the encrypted_hash received is decrypted using sender's public key, if this hash matches with the hash of the data, we conclude that the message is authentic. Otherwise, the message is rejected.

11) Once the hash is confirmed the message decryption can go ahead at receiver's end, padded bytes are first removed.

12) Then the encrypted message list is decrypted and finally converted back to the original format.

Theorem: If the RSA problem is hard relative to GenRSA then the above algorithm has indistinguishable encryptions under chosen-plaintext attacks.